NOTICE CONCERNING COPYRIGHT RESTRICTIONS

The copyright law of the United States [Title 17, United States Code] governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that use may be liable for copyright infringement.

The institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law. No further reproduction and distribution of this copy is permitted by transmission or any other means.
value of \( \lambda \), we solve for. If the \( \sigma_i \) are the roots of \( \mu_i \), this may not always have to be considered of \( i \), then \( 2^k \)
which satisfy \( 3.14 \)

4.3

the value for \( \lambda \) for the \( \mu_i \) then

st (i.e. in the curve goes through e all conditions, then gives the limits be improved by use or by the numerical

5.

THE BRADFORD METHOD FOR THE ASSIGNMENT PROBLEM

by C. Mack, Institute of Technology, Bradford.

Summary: This method is at once simpler and faster than any previously propounded method. It can be taught to unskilled personnel (the ability to add and subtract is needed), at the same time it needs less and much easier programming, is much quicker, and is always certain of success when used on digital computers.

Also given is a proof that such methods as have been proposed for the 2-dimensional problem will not succeed with the 3-dimensional allocation problem. Finally, methods of finding the 2nd, 3rd, etc., best solution for the 2-dimensional problem are given.

1. INTRODUCTION

The assignment (or 'allocation' or 'optimal assignment') problem is this: Given an \( n \times n \) square array of numbers \( a_{ij} \), define a 'possible' set as the set

\[
\{ a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n} \}
\]

where \( k_1, k_2, \ldots, k_n \) is a permutation of \( 1, 2, \ldots, n \). Then find that possible set which has the minimum sum (or in some cases the maximum sum). We shall call such sets 'minimal' or 'maximal'. Note that the set \( 1.1 \) consists of one element from each row but that no two of them are in the same column (hence, since there are \( n \) of them they must be in separate columns). An example of a possible set is the set of diagonal elements i.e. \( a_{11}, a_{22}, \ldots, a_{nn} \).

The assignment problem may arise in many practical situations. One simple example is the allocation of \( n \) jobs to \( n \) workers whose rate of production for each job is known. The optimal allocation gives the maximum rate of production. The problem has arisen in the radar systems now being developed to control the large numbers of aircraft expected to be flying simultaneously within a few years. The Royal Radar Establishment has, after careful consideration, adopted the Bradford method (see Magowan, 1965). Other organisations are adopting it (or so I believe).
Easterfield (1946 and 1961) put forward a method which the Bradford method resembles in many respects, but the latter is very much quicker. Kuhn (1955) gave a 'Hungarian' method based on existence theorems by Hungarian mathematicians. However, his method is only practicable on an electronic digital computer and requires skilful programming besides taking considerable computer time. Churchman, Ackoff, and Arnoff give a variant of the Hungarian method which is reasonably good for small arrays, but the problem of finding the least number of lines through the zeroes can be quite difficult even for a 5x6 array. (Kuhn's method, in effect, classifies these zeroes in several different classes, finally sorting them out and thus determining the minimum number of lines, but the zeroes have to be examined quite a number of times in this process.)

The Bradford method has none of these disadvantages and the technique described below can be taught to unskilled clerks in half an hour. It or its variant of Section 3 can be easily programmed (full instructions are given in Section 3), and it requires relatively little programming, takes little computer time, and is always certain of success.

The desk form of the method given in Section 2 has a simple check (most important for desk work) and with it a 15x15 array can be done in less (sometimes much less) than an hour.

All methods, directly or indirectly, are based on the observation that, if we add to or subtract from every element in a row (or column) the same amount, we do not alter the positions of the minimal or maximal set. How this is used in the Bradford method we now describe.

2. THE BRADFORD METHOD

2.1 The nature of the basic operations. We shall describe the case of finding the minimal set (the finding of the maximal set is described in subsection 2.6). Now in every row there is a minimum element (for that row); though in some rows there may be alternative equal minimum elements. In the Bradford method we raise (by a single column, or on some of the rows, or on some of the rows) we can select one from each row such that different columns. one such element in form a minimal set, consists of one element $\geq$ the sum of the rows in the same positions as the required answer.

This raising which are now described.

2.2 The operations and an 'alternative', underline in each row any chosen arbitrarily element. Each underlined element can be seen in array. Only one base in each column and no bases only occupy 3 columns.

(a) Start of Step I

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>6</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>9</td>
<td>18</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>14</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>14</td>
<td>21</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
<td>10</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>14</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

We spread to each 'step' by the with more than one (every element in) of these bases equal but in the same row 'alternative base' to satisfy this we place a
method we raise (by the same amount) the elements of a single column, or of several columns, so that we form two or possibly more alternative minimum elements in some of the rows. We keep doing this until finally we can select one from the alternative minimum elements in each row such that these chosen elements are in different columns. There must then be one and only one such element in each column. These elements must form a minimal set because any other possible set consists of one element from each row and so its sum is the sum of the row minima. Finally, the elements in the same positions in the original array give the required answer.

This raising of columns is done in 'steps' which are now described.

2.2 The operations in a 'step': definition of a 'base' and an 'alternative base'. If it is the first step underline in each row the minimum element (or one chosen arbitrarily if there are two or more). We call each underlined element a 'base'. This underlining can be seen in array (a). Note that there is one and only one base in each row but there may be several in some columns and none in others as in (a) where the bases only occupy 3 columns.

(a)

<table>
<thead>
<tr>
<th>Start of Step 1</th>
<th>End of Step 1 = Start of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 12 6 25 22 20</td>
<td>4 12 6 25 22 20</td>
</tr>
<tr>
<td>5 18 9 18 25 27</td>
<td>5 18 9 18 25 27</td>
</tr>
<tr>
<td>5 13 14 21 29 28</td>
<td>6 13 14 21 29 28</td>
</tr>
<tr>
<td>3 8 14 21 25 24</td>
<td>11 8 14 21 25 24</td>
</tr>
<tr>
<td>11 16 10 22 14 25</td>
<td>14 16 10 22 14 25</td>
</tr>
<tr>
<td>2 21 14 12 16 15</td>
<td>12 21 14 12 16 15</td>
</tr>
</tbody>
</table>

We spread the bases over one more column at each 'step' by the following technique: Take a column with more than one base (e.g. col 1 in (a)) and raise (every element in it) by the least amount so that one of these bases equals an element in another column but in the same row. This other element is then an 'alternative base' to the original base, and to signify this we place a dot under it. Thus if we raise...
col 1 of (a) by 3, (when (a) becomes (b)), the original base 9 in row 6 col 1 becomes 12 and equals the 12 in row 6 col 4 (so we place a dot beneath it). Since this alternative base is in a column (col 4) containing no original bases, we stop. We then transfer the base in row 6 from its original position (col 1) to the alternative position (col 4) and thus we have spread the bases over one more column than before. The new base is underlined and the line under the original base in erased (or brackets put around it). The bases then appear as in (b) where they occupy 4 columns as against only 3 in (a). Step 1 has now ended.

The end of any previous 'step' is the start of the next 'step'. Thus we start step 2 with the array (b). Again, we raise col 1 (it has 3 bases), this time by 2, when the base 4 in row 1 becomes 6 and equals the 6 in row 1 col 3. This 6 is now an alternative base and is dotted (see (b); but we do not show the raised col 1 in (b)). This time, unfortunately, the new alternative base is in a column (col 3) already containing an original base (namely 10 in row 5). So now we raise both col 1 and col 3 by 4, the base in row 5 col 3 now goes from 10 to 14 and equals the element 14 in row 5 col 5 (it is therefore dotted). Since col 5 has no original base we stop. The array is now as in (c), and we are ready to transfer the bases. (d)

<table>
<thead>
<tr>
<th>Before Transfer</th>
<th>Diagram showing Transfer of Bases</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 12 10 25 22 20</td>
<td>(X) x x x x x</td>
</tr>
<tr>
<td>11 13 18 13 25 27</td>
<td>X x x x x x</td>
</tr>
<tr>
<td>12 13 18 21 29 28</td>
<td>X x x x x x</td>
</tr>
<tr>
<td>17 18 21 25 22 24</td>
<td>X X X X X</td>
</tr>
<tr>
<td>20 16 14 22 14 25</td>
<td>X X (X) X X</td>
</tr>
<tr>
<td>18 21 18 12 16 15</td>
<td>X X X X</td>
</tr>
</tbody>
</table>

This transfer of bases so as to occupy one more column is shown diagrammatically in (d). Note that in col 3 a base is removed (in row 5) but a new one is formed (in row 1); while col 5 has a new base, and in col 1 one base is removed.

In general a 'step' consists of a number of raising operations. In such an operation the starting column plus all columns formed previously do not contain a least amount required. We stop when the last column becomes original. We then occupy one more column.

2.3 The Transfer Rule (though it is very easy to follow)

The last formed alternative base is an alternative base in the original "column" and so on; until a column contains other (originals). This success in a column which hase been formed from a column which has already interconnected and so on. Note that raising will necessarily appear in (g) below; (ii) so we have alternative bases before the above process (this is column 1).

2.4 The Short Form to add to elements of a column (because addition only)

So the following She gives the order of addition clear. The array (c) going from (b) to (d) operations clear. In raising all that are raised in raising by raising column 1. The bases of column 1 in A is the amount of bases then become 6 in column 6 in row 6.
s (b)), the original
and equals the 12 in
ath it). Since
n (col 4) containing
en transfer the base
. (col 1) to the
us we have spread
before. The new
ler the original base
). The bases then
 columns as against
ed.
step' is the start of
tep 2 with the array
as 3 bases), this time
omes 6 and equals the
an alternative base
ot show the raised
unately, the new
ol 3) already contain-
row 5). So now we
the base in row 5
quals the element 14
itten). Since col 5
y array is now as in
the bases.

ing Bases

<table>
<thead>
<tr>
<th>x</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

(_) means a
removed base

x x means a
new base

as to occupy one more
in (d). Note that in
) but a new one is
a new base, and in

2.3 The Transfer Rule We now give this in words,
though it is very easily demonstrated graphically (see
(d) above). It is

The last formed alternative base is always a new base
the original base in the above element's row is removed
an alternative " " " " " . \( \text{col is a new base} \)
the original \( \text{row is removed} \)
an alternative " " " " " \( \text{col is a new base} \)
and so on; until a base is removed from a column which
contains other (original) bases.

This succeeds because (i) a new base is formed
in a column which had none before; (ii) a base is removed
from a column which had more than one; (iii) columns
intermediately have a base removed from one row but have
a new base in another row.

Note that not all columns carrying alternatives
will necessarily appear in the above transfer (see array
(g) below); (ii) some columns may carry two or more
alternative bases but only one is selected during the
above process (this can be done arbitrarily, see (k)).

2.4 The Short Form Now there is no need, during a step,
to add to elements other than the bases in a column
(because addition cannot make a non-minimum a minimum).
So the following Short Form has been devised which (i)
confines addition to essential numbers only, (ii) sets
out all operations simply and clearly.

The array (e) below shows the above Step 2 (i.e.
going from (b) to (c)) in Short Form and makes its oper-
ations clear. In (e) the line C gives the columns
that are raised in chronological order. Thus we start
by raising col 1. Below the 1 (in line C) we place
the bases of col 1 (namely 4, 5, 6). The first amount
in A is the amount by which col 1 is raised. The
bases then become 5, 7, 8 as shown. But the 6 equals
the element 6 in row 1 col 3, so a dot is placed under
both these elements. Since the alternative base is in col 3 then 3 is inserted in line C and the base in col 3 (namely 10 in row 5) is added in the appropriate place below the 3 (in line C). The second amount 4 in line A is that by which both col 1 and col 3 are raised. Their bases now become 10, 11, 12 and 14 (see the last column on the right in (e)). The last element equals the 14 in row 5 col 5 and a dot is placed under both. Since col 5 has no (original) base we stop.

The transfer of bases can then be marked in very simply (it is shown in (f) above, to avoid too much marking cf (e), but it would normally be marked on (e)). After this marking of the transfer, the amount T nominally added to each element of a column during the complete step is tallied and actually added to that column. The rule is this: $T = \text{Total amount shown in } \text{line A to the right of where the column is mentioned in line } C$; (thus, for col 3, $T = 4$; for col 1, $T = 4 + 2 = 6$). The bases are then marked in using the markings on the right, and the array is ready to start the next step, (the array resulting from (e) is shown in (g)).

Further shortening of the process. If the working is carried out in pencil then the augmented columns which arise from the addition of the T can be written (with the correct bases underlined) next to their original columns and the latter erased. When this has been done the working on the right is erased. This saves having to write out complete arrays (a disadvantage of the Hungarian method).

Further suggestions:

(i) prime (or small) elements not being raised (smallest elements are being raised; would be confusing)

(ii) the minimum element and the column on the right gives the elements 20, 18, 21, under the 3 in line A. A value is min(20, 18, 21)

(iii) draw a vertical line (number appears in a glance)

(iv) where and through a primed element in another column

By the full 20 x 20 arrays can be
Further simple but valuable time-saving devices are:

(i) prime (or tick), in each row of the bases of the columns being raised, the smallest element in a column not being raised (in (g) below, we show the primed 'smallest elements' at the point where cols 1, 2, and 3 are being raised; the full priming throughout the step would be confusing to show)

(ii) the minimum difference between primed 'smallest element' and the corresponding (augmented) base value on the right gives the A value (thus, with the primed elements 20, 18, 21, 21 shown in (g) the augmented bases under the 3 in line C are 10, 13, 14, 9, and so the next A value is \( \min\{20-10, 18-13, 21-14, 21-9\} = 5 \)

(iii) draw a vertical line through any column whose number appears in line C (columns being raised can then be seen at a glance)

(iv) where and only where this vertical line passes through a primed element then another 'smallest element' in another column must be found.

By the full short technique any \( 15 \times 15 \) and some \( 20 \times 20 \) arrays can be solved within an hour.

\[ (g) \]

<table>
<thead>
<tr>
<th>Step 3 in Short Form</th>
<th>A: 1</th>
<th>1</th>
<th>1</th>
<th>5</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>25</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>18</td>
<td>21</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>8</td>
<td>18</td>
<td>21</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>16</td>
<td>14</td>
<td>22</td>
<td>14</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>21</td>
<td>18</td>
<td>12</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>:T</td>
</tr>
</tbody>
</table>

The array (g) shows in Short Form, the last step, for our illustrative array with the last transfer of bases shown on the right. When the T values are added to their columns we obtain the array (h). Actually, there is no need to add the final step T values because we are only interested in the positions of the bases (and (j) shows the minimal set for the original array). However, the final T-values are themselves of interest in that they enable a simple check to be carried out.
Further simple but valuable time-saving devices are: (i) prime (or tick), in each row of the bases of the columns being raised, the smallest element in a column not being raised (in (g) below, we show the primed 'smallest elements' at the point where cols 1, 2, and 3 are being raised; the full priming throughout the step would be confusing to show).

(ii) the minimum difference between primed 'smallest element' and the corresponding (augmented) base value on the right gives the A value (thus, with the primed elements 20, 18, 21, 21 shown in (g) the augmented bases under the 3 in line C are 10, 13, 14, 9, and so the next A value is min{20−10, 18−13, 21−14, 21−9} = 5).

(iii) draw a vertical line through any column whose number appears in line C (columns being raised can then be seen at a glance).

(iv) where and only where this vertical line passes through a primed element then another 'smallest element' in another column must be found.

By the full short technique any 15×15 and some 20×20 arrays can be solved within an hour.

---

In short form, the array (g) shows the last step for our illustrative array with the last transfer of bases shown on the right. When the T values are added to their columns we obtain the array (h). Actually, there is no need to add the final step T values because we are only interested in the positions of the bases (and (j) shows the minimal set for the original array). However, the final T-values are themselves of interest in that they enable a simple check to be carried out.
(h) Final (and Check) Array

20 21 18 25 22 20
21 27 21 24 25 27
22 22 26 24 29 28
27 17 26 24 25 24
30 25 22 25 14 25
28 30 26 15 16 15

W: 19 9 12 3 0 0

It is to be noted that in the transfer of bases shown in
(g), only 2 out of the 4 alternative bases (shown dotted
on the right) appear as new bases. Note also that in
the minimal set shown in (j) the two smallest elements
1 and 2 of the array do not appear.

2.6 The Check The line W in (h) gives the sum of the
T values added to the columns in the whole process.
If these W values are added to the original columns
then we should get the final array. If the row minimum
in this final array are in the positions given by the
final step we have made no mistake (not in finding the
minimal set; there might be other mistakes but these do
not matter).

2.7 The Maximal Set This can be found in exactly the
same way as the minimal set except that we underline
the row maximum as a base; and in a step we lower those
columns which have more than one base to form alternative
bases.

In (k) below we show a single step in Short Form.

(j) Original Array showing Minimal Array Set

1 12 6 25 22 20
2 18 9 18 25 27
3 13 14 21 29 28
4 8 14 21 25 24
11 15 10 22 14 25
9 21 14 12 16 15

3. ALTERNATIVE FORM

3.1 Basic Operation of the machine was programmed without
columns being raised or lowered, etc. This method is perhaps
more efficient. In this form we operate on columns by
raising one element only of each column by a single
value, which is added to the original column. In this way
each step by selecting columns we lower it by the
elements to equal the bottom base of the column.

3.2 The Transfer Rule

The (original) base becomes an alternative base of the
original "column" and so on until a new base is
found at the start of the new column.

This succeeds column containing the bases removed, into
the base, and the first
(i) Final Array showing Minimal Set

<table>
<thead>
<tr>
<th>6</th>
<th>25</th>
<th>22</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>18</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>21</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>21</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>14</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>

There are some features worth noting in (k). (i) When col 3 is brought in, it has two original bases and care must be taken to ensure that both are recorded below the 3 in line 6 on the right. (ii) In the last but one column on the right two alternative bases have been formed simultaneously (16 and 20); during the transfer of bases one and one only of these has to be selected. We do this arbitrarily, since there is no means of telling beforehand which will shorten the subsequent work.

3. ALTERNATIVE FORM OF BRADFORD METHOD: PROGRAMMING

3.1 Basic Operations The previous method can be programmed without difficulty; but, since addition to the columns being raised presents no difficulty on an electronic computer, we now give an alternative form, which is perhaps more easily programmed.

In this form, to find the minimal set, we start each step by selecting a column containing no bases, and we lower it by the least amount for one of the column's elements to equal a base. This element becomes an alternative base. If the (original) base equalled is in a column containing other bases we stop; if not we lower this and the first column to form another alternative base; and so on. (The result of carrying out such a step on the array (b) is shown in (m) below.) Just as in the previous method we have to transfer the bases to occupy one more column which we do thus:

3.2 The Transfer Rule The (original) base last equalled is removed an alternative base in the above's row is a new base the original " " " col " removed an alternative " " " row " a new base and so on.

until a new base is formed in the column first lowered at the start of the step.

This succeeds, just as in 2.3, because the column containing the last equalled base has one of its bases removed, intermediate columns lose one and gain one base, and the first column lowered gains one.
3.3 The Programme. We need the following quantities during the programme (which is for the minimal set):

- \( a_{ij} \): current value of the element in the \( i \)th row and \( j \)th column of the array.
- \( n_j \): the number of bases in the \( j \)th column.
- \( b_i \): the value of the base in the \( i \)th row, and
- \( c_i \): the number \((\text{i.e. the } j\text{-value})\) of its column.
- \( a_i \): the number of the column containing the alternative base in the \( i \)th row \(\text{ (if none, put } a_i = 0)\).
- \( w_1, w_2, \ldots, w_k \): the numbers of the columns being lowered during the \( k \)th lowering operation of a step.

We commence by finding \( b_i \) and \( c_i \) for the original array, and from the \( c_i \) we find the \( n_j \). We are then ready for step 1. A step is started thus:

1) put all \( a_i = 0 \), and put \( w_i = j \) first \( j \) whose \( n_j = 0 \). Then we begin the lowering operations. At the \( k \)th such operation:

2) find \( M = \min(a_{ij} - b_i) \) for all \( i \) whose \( a_i = 0 \) and for \( j = w_1, \ldots, w_k \); suppose \( i = i', j = j' \) at this minimum.
3) put \( a_{i'} = j' \); subtract \( M \) from all \( a_{ij} \) for \( j = w_1, \ldots, w_k \).
4) if \( n_{c_{i'}} = 1 \), put \( w_{k+1} = c_{i'} \), \( k = k+1 \), jump to 2).
5) if \( n_{c_{i'}} > 1 \), subtract 1 from \( n_{c_{i'}} \), put \( n_{c_{i'}} = 1 \).

Then effect the transfer of bases thus: put \( i = \text{last } i' \), \( c_{i'} = i \) at the \( t \)th transfer.
6) put \( c_{i+1} = a_{i+1} \); if \( a_i = J \) jump to 7), if not
- find \( i_{t+1} \) such that \( a_{i_{t+1}} = a_i \); put \( t = t+1 \), jump to 6).
7) set all \( b_i = a_{i+1} \) where \( j'' \) = (new) \( a_i \); jump to 1)

When no \( n_j = 0 \) \(\text{ (when all } n_j \text{ should be 1)}\), the \( c_i \) = column number of the element of the minimal set in the \( i \)th row.

The programme for the maximal set is much the same, though then the \( b_i \) are the maxima in their rows, \( M = \min(b_{ij} - a_{ij}) \), and \( M \) is added not subtracted in 3).

We now show some of the details of the working in carrying out a step by the above programme, starting with the array of

A Step of the Alternate Values of the
Starting Value

<table>
<thead>
<tr>
<th>i or j</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_j )</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( b_i )</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>( c_i )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( a_i )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Before Transfer

| \( n_j \) | 2 | 1 | 1 |
| \( a_i \) | 3 | 0 | 0 |

After Transfer

| \( c_i \) | 3 | 1 | 1 |
| \( b_i \) | 4 | 5 | 6 |

Final Result

| \( a_i \) | 4 | 12 | 4 |
| \( b_i \) | 5 | 18 | 7 |
| \( c_i \) | 6 | 13 | 2 |
| \( n_j \) | 11 | 8 | 12 |
| \( a_i \) | 14 | 16 | 8 |
| \( b_i \) | 12 | 21 | 12 |

4. The 3-Dimension

We shall show of raising or lowering...
wing quantities

i the ith row and

(column

th row, and

ating the alternative

put a_i = 0)

olumns being lowered

ion of a step

r the original array;
e are then ready for

st j whose n_j = 0

ns. At the kth

whose a_i = 0 and for

at this minimum

l a_j for j = w_1, ..., w_k

k+1, jump to 2)

, put n_j = 1,

thus: put i = last i',

to 7), if not

; put t = t + 1, jump to 6)

ew) a_i ; jump to 1)

de 1), the c_i =

minimal set in the

al set is much the

axima in their rows,

ot subtracted in

tails of the working

ue programme, starting

ith the array of (b) given earlier.

A Step of the Alternative Method (applied to (b))

<table>
<thead>
<tr>
<th>Starting Values</th>
<th>During Lowering</th>
</tr>
</thead>
<tbody>
<tr>
<td>i or j</td>
<td>1</td>
</tr>
<tr>
<td>nj</td>
<td>3</td>
</tr>
<tr>
<td>bj</td>
<td>4</td>
</tr>
<tr>
<td>ci</td>
<td>1</td>
</tr>
<tr>
<td>ai</td>
<td>0</td>
</tr>
<tr>
<td>J = 5</td>
<td>4)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Before Transfer

| nj | 2 | 1 | 1 | 1 | 0 | |
| a_i | 3 | 0 | 0 | 0 | 5 | 5 |

Transfer

| t | 1 | 2 |

After Transfer

| ci | 3 | 1 | 1 | 2 | 5 | 4 |
| bj | 4 | 5 | 6 | 8 | 8 | 10 |

(\textit{m})

Final Result of the above Step

\[
\begin{array}{cccccc}
4 & 12 & 4 & 23 & 16 & 20 \\
5 & 18 & 7 & 16 & 19 & 27 \\
6 & 13 & 12 & 19 & 23 & 28 \\
11 & 8 & 12 & 19 & 19 & 24 \\
14 & 16 & 8 & 20 & 8 & 25 \\
12 & 21 & 12 & 10 & 10 & 15 \\
\end{array}
\]

4. THE 3-DIMENSIONAL ALLOCATION PROBLEM

We shall show by an example that the technique
of raising or lowering columns, rows, etc., will not
succeed, in general, here.

The general element in an array here is $a_{ijk}$.
The 2-dimensional array formed when $i$ = constant we shall
call a row; when $j$ = constant, a column; and when $k$
constant a 'level'. Consider the following $2\times 2\times 2$ array:

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>0</td>
</tr>
<tr>
<td>Row 2</td>
<td>1</td>
</tr>
</tbody>
</table>

The minimal set is 0, 0 as trial and error will show.
We shall show that by adding $r_1, r_2$ to the rows and $c_1, c_2$
to the columns we cannot make the elements in the above
$eta, \gamma$'s positions simultaneously zero. The modified array is:

\[
\begin{array}{cc}
4+c_1 & r_1+c_1+1 \\
2+c_2 & \ \ \ 2+c_1+10 \\
\end{array}
\]

If the underlined elements are minimax in their levels,
then it follows that

\[
\begin{align*}
c_2-c_1-4 & > 0 \quad (1) \\
r_2-r_1+c_2-c_1-4 & > 0 \quad (2) \\
c_1-c_2+10 & > 0 \quad (4) \\
r_1-r_2+c_1-c_2+10 & > 0 \quad (5) \\
r_1-r_2+1 & > 0 \quad (6)
\end{align*}
\]

From (1) and (5), $r_1-r_2 \geq c_2-c_1-1 > 4 - 1 = 3$

From (3), $1 > r_1-r_2$

and the results are incompatible.

Similarly, adding to the rows and levels will
not make the underlined elements minima in their columns,
and adding to the levels and columns will not be success-
ful either.

An intuitive way of seeing that such methods
are unlikely to be successful is this: In the 2-dimen-
sional array we have $n^2$ elements and (in effect) 2n
quantities we can add, $n$ to the columns and $n$ to the rows.
In the 3-dimensional case we have $n^3$ elements and only
3n quantities we can add. This is an order of magnitude
too small.

5. THE 2ND, 3RD, ETC. BEST 2-DIMENSIONAL SOLUTION

However, it is possible to provide answers to
these problems. We can find the 2nd best minimal
solution this way: Take the minimal set, and replace
each element of this set in turn by a large number

(say the sum of the
set for each of the
set is the 2nd best
only slightly might be the
possible for the set element with the larger

\[
\begin{array}{cc}
0 & 100 \\
1 & 1
\end{array}
\]

the best solution.

The 3rd best minimal set for the
common element of
(i) making a non-
common element of
The smallest of these
in an exceptional
found. The situation
can be found along

REF.

CHURCHMAN, C.W.,
Introduction to

EASTERTFIELD, T.R.
J. Lond. & (1961) 'An
Operat. Res.

KUHN, H.W., (1955)
ment Problem

MAGOWAN, S., (1965)
Monog.
Here is an example of a 2x2x2 array:

\[
\begin{array}{ccc}
1 & 0 & 0 \\
0 & 2 & 1 \\
\end{array}
\]

If we allow 2x2x2 arrays and when \( k = \min \), the error will show on the rows and \( c_1, c_2 \) elements in the above.

The modified array:

\[
\begin{align*}
r_1 + c_1 + 1 \\
r_2 + c_2 \\
\end{align*}
\]

in their levels,

\[
+ \geq 0 \quad (2) \quad r_2 - r_1 + 1 > 0 \quad (3) \\
+ \geq 0 \quad (5) \quad r_1 - r_2 + 1 > 0 \quad (6)
\]

\( 4 - 1 = 3 \)

ows and levels will nina in their columns, so will not be success that such methods is: In the 2-dim- and (in effect) 2n yms and n to the rows. \( 3 \) elements and only an order of magnitude

IONAL SOLUTION provide answers to 2nd best minimal set, and replace a large number

(say the sum of the row maxima). We find the minimal set for each of these modified arrays and their minimal set is the 2nd best solution. This will not take so long as might be thought, for the arrays will differ only slightly from the original array. However, it is possible for the second-best solution to have no common element with the best solution. In this array

\[
\begin{array}{ccc}
0 & 1 & 100 \\
1 & 0 & 1 \\
100 & 0 & 0 \\
\end{array}
\]

the best solution is 0,0,0 and the 2nd best 1,1,1.

The 3rd best solution is found by finding the minimal set for the arrays formed by (i) making any common element of 1st and 2nd best solutions very large, (ii) making a non-common element of the 1st and a non-common element of the 2nd simultaneously very large. The smallest of these solutions is the 3rd best. But, in an exceptional case \( n^c_3 \) minimal sets might have to be found. The situation is worse for the 4th best which can be found along similar lines.

REFERENCES


-29-